Lecture 3 Review

C++ basic program structure:

```cpp
#include<iostream>… (“header files” are essential)

// comment indicator (stuff to right on same line ignored)

std::cout , std::cin are the std output, input

int main () (all programs need a main function)

g++ compiler (source code → executable code)
```


!! Scream if you get stuck !!
Number Representation on a Computer

- “Computers are not infinitely precise in their calculations.”
- We need to pay attention to significant figures. (As in lab!!)
- Real numbers represented in binary form: fixed-point or floating point
  - **Fixed point** (fixed number of digits before/after decimal point.)
- N bits used to represent number I (e.g., 23.45)

\[ I = \text{sign} \times (\alpha_n2^n + \alpha_{n-1}2^{n-1} + \ldots \alpha_0 + \ldots \alpha_m2^{-m}) \]

with \( n + m = N - 2 \) and N, m, n machine dependent

**Advantage:** All FxP numbers have same absolute error: \( 2^{-m-1} \)
- Can represent fractional powers of 2 exactly.

**Disadvantage:** Cannot represent exactly fractional powers of 10.

- We won’t use FxP numbers all that much.
Number Representation on a Computer

We will use “floating point numbers:” use a representation of a number where the decimal can float around wrt sig figs and then adjust matters via an exponent. Think scientific notation.

**Advantage:** Greater range of numbers can be represented wrt FxP rep.

- We’ll use floating point rep for numbers almost exclusively.

\[ x_{float} = (-1)^s \times 1.f \times 2^{e-bias} \]

\( s = \) sign bit. \( f = \) mantissa \( e = \) “exponent field” \( bias = 127_{10} \)

“real” exponent = \( p = e - bias \) (always want \( e \geq 0, \forall p \))

<table>
<thead>
<tr>
<th>Bit position</th>
<th>s</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>23</td>
<td>22</td>
</tr>
</tbody>
</table>

Assumption: 4 “bytes” = 32 bits used to store number.
Floating Point Representation of a Number

<table>
<thead>
<tr>
<th>Bit position</th>
<th>s</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>23</td>
<td>22</td>
</tr>
</tbody>
</table>

\[ \text{mantissa} = 1.f = 1 + m_{22} \times 2^{-1} + m_{21} \times 2^{-2} + \cdots + m_0 \times 2^{-23} \]

23 bits used to set precision of number (IF 4 bytes used, you decide.) precision = 1 part in \(2^{23}\). What is this in plain English? Hint: \(2^{10} = 1024\) (call it an even 1000 for estimation purposes).

Q: What the is \(2^{23}\)? And then \(1/2^{23}\)?

This ratio sets the limit on the precision your computer recognizes, regardless of exponent: e.g., \(1.00000005 \times 10^{-22} = 1.0 \times 10^{-22}\)

(IF using 32 bits to store a number. We will verify on our machines.)
Floating Point Representation of a Number (2)

<table>
<thead>
<tr>
<th>Bit position</th>
<th>s</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Range of “exponent field e”: $0 \leq e \leq 255$ (Note: 256 values = $2^{^8}$.)

Jargon: “normal floating point number”: $0 < e < 255$

Q: What is largest positive normal fp number? (Yes, a question to you !)

Recall: $x_{float} = (-1)^s \times 1.f \times 2^{e-bias}$

mantissa = $1.f = 1 + m_{22} \times 2^{-1} + m_{21} \times 2^{-2} + \cdots + m_0 \times 2^{-23}$

$1.f = 1.1111\ 1111\ 1111\ 1111\ 1111\ 1111$\ 111

$p = e - \text{bias} = e_{10} - 127$ (p is the “real” exponent you want)

Answer = ?????
“Double Precision” Numbers

Typically require more precision than just 32 bit representation.

Solution: Use 2 X 32 bits = 64 bit representation. (Who knew?)

Very simple to do in C++ (and other languages). See how soon.

<table>
<thead>
<tr>
<th>Bit position</th>
<th>s</th>
<th>e</th>
<th>f</th>
<th>f (cont.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>63</td>
<td>62</td>
<td>52</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td>31 0</td>
</tr>
</tbody>
</table>

HW problem: Estimate precision for such “double precision” numbers.
Reference: See CP, sec 2.5 -2.7.

Always use double precision numbers for scientific computing.
```
#include <iostream>

using std::cout; using std::cin; using std::endl;

int main()
{
    float one = 1;
    float eps = 0.02;
    int N;
    cout << " N = " ;
    cin >> N;
    cout << "N = " << N << endl;

    for (int i = 0; i < N; ++i){
        eps = eps/2.;
        one = 1. + eps;
        cout << "one =  " << one << " 	 step = " << i << " 	 eps = " << eps << endl;
    }

    return 0;
}
```

Even w/ double precision (64 bits), computer precision is **not** infinite.

\[ x_c = x(1 \pm e) \quad \text{w/ } |e| \leq e_m. \]

How to measure \( e_m \)?
Execute Machine Precision Code

Edit and compile previous program:

```sh
g++ -o mach_precision mach_precision.cc
```

Q: What is N?
Q: What is \( e_m \)?

**Useful** linux trick: Put interactive executable in shell “script.”

```sh
#!/bin/tcsh –f
mach_precision << stuff
```

### Place in file

30 input

- **Req’d magic symbol**
- **Must match**

Troubles getting your script to run? First, `ls -l your_file`

Use `cx` to make script file executable. Try `which cx`
Help with C++ Variables and For Loop

My head is exploding.

I need something to read quietly, at my own pace.

http://www.cplusplus.com/doc/tutorial/variables.html
http://www.cplusplus.com/doc/tutorial/control.html

Link also available from PHYS 3340 links page
Summary

- Representation of single & double precision real numbers.
- Either representation has a finite precision.
- Code to determine machine precision for single precision numbers.
- Example of variable declaration (single precision real).
- Example of for loop.
- Simple example of a “here document” in shell scripting.

You should have finished linux tutorial

Don’t suffer in silence. Scream for help!!!